EL575422638US

SUN-P4175

This application is submitted in the name of inventor Judith E. Schwabe, assignor to Sun Microsystems, Inc., a Delaware Corporation.

S P E C I F I C A T I O N

API REPRESENTATION ENABLING SUBMERGED HIERARCHY

BACKGROUND OF THE INVENTION

Cross Reference to Related Applications

This application claims the benefit of provisional patent application Serial No. 60/165,298 filed November 12, 1999 in the name of inventor Judith E. Schwabe, entitled "API Representation Enabling Submerged Hierarchy". This application also claims the benefit of provisional patent application Serial No. 60/165,533 filed November 15, 1999 in the name of inventor Judith E. Schwabe, entitled "API Representation Enabling Submerged Hierarchy".

This application is related to the following:

U.S. Patent Application filed September 14, 2000 in the name of inventor Judith E. Schwabe, entitled "Remote Incremental Program Verification Using API Definitions", Attorney Docket No. SUN-P4172, commonly assigned herewith.

U.S. Patent Application filed September 14, 2000 in the name of inventor Judith E. Schwabe, entitled "Remote Incremental Program Binary Compatibility Verification Using API Definitions", Attorney Docket No. SUN-P4174, commonly assigned herewith.

U.S. Patent Application filed September 14, 2000 in the name of inventor Judith E. Schwabe, entitled "Populating Resource-Constrained Devices With Content Verified Using API Definitions", Attorney Docket No. SUN-P4176, commonly assigned herewith.

U.S. Patent Application filed September 14, 2000 in the name of inventor Judith E. Schwabe, entitled "Populating Binary Compatible Resource-Constrained Devices With Content Verified Using API Definitions", Attorney Docket No. SUN-P4182, commonly assigned herewith.

U.S. Patent Application Serial No. 09/243,108 filed February 2, 1999 in the name of inventors Judith E. Schwabe and Joshua B. Susser, entitled "Token-based Linking".

Field Of the Invention

The present invention relates to computer systems. More particularly, the present invention relates to a method for representing an application programming interface (API) in an object-oriented system such that submerged hierarchies are enabled.

Background

An API definition file typically defines a library of functionality that may be used by one or more client applications in an object-oriented system. An API definition file also typically includes the set of classes and interfaces that are externally accessible from that library and the items in those classes and interfaces that are externally accessible. A library in Java™ technology corresponds to the Java™ "Package" construct.

The "items" in classes and interfaces include fields, methods and implemented interfaces of the various classes, and fields and methods of the various interfaces. Additionally, the immediate superclass and superinterface for each class and interface, respectively, is listed. Since an API can only enumerate externally accessible items, all superclasses and superinterfaces must also be externally accessible.

An API definition file may include non-public functionality that is not exposed by the API definition file. This functionality can be implemented in classes as non-public fields, non-public methods or non-public implemented interfaces. In addition, the content of a method is not disclosed in an API representation, regardless of whether the method is public or non-public. Therefore, non-public algorithms are not disclosed.

Figure 1A illustrates a typical API representation. In this example, all of the classes in the hierarchy of C2 (2) are public. The API representation indicates that the public class C1 (4) is the superclass of the public class C2 (2). The API representation also indicates that the public class Object (6) is the superclass of the public class C1 (4). Thus, this API representation reveals the entire hierarchy of C2 (2). Furthermore, typical API representations unnecessarily constrain a hierarchy by forcing the hierarchy to contain only public classes.

Additionally, since typical API representations only indicate the immediate superclass or superinterface of a particular class or interface, respectively, one must traverse the hierarchy recursively to determine whether a class or interface is a member of the hierarchy. Such functionality requires that not only the immediate API definition file is available when validating references to elements in a hierarchy, but also the set of API definition files referenced by that API definition file.

The Java™ language supports a construct where the immediate superclass or superinterface of a class or interface, respectively, may be declared as non-public. A superclass or superinterface declared in this way must not be disclosed in an API definition file. Figure 1B demonstrates this feature in a class hierarchy. Such a hierarchy that includes one or more non-public classes is referred to as a submerged hierarchy. Such constructs are useful to API designers and implementers because they allow non-public, or proprietary functionality to be inserted into a hierarchy and for that functionality to be encapsulated in a hidden class. This provides for modular designs consistent with Object-oriented theory.

However, the hierarchy illustrated in Fig. 1B cannot be represented in a typical API definition file. This is because a typical API representation requires the disclosure of the immediate superclass of C2 (8), which is PrivateClass (10). Disclosing a non-public class such as PrivateClass (10) would violate the Java™ language requirement that such a class not be made public.

Typical API representations constrain the design of an API such that more than is desirable falls into the publicly accessible domain. Accordingly, a need exists in the prior art for an API representation that sufficiently constrains particular implementations, while allowing them to define submerged hierarchies.

SUMMARY OF THE INVENTION

A method for representing an application programming interface (API) for an object-oriented library includes creating a list of public elements in the library and storing the list. Each public element in the list includes a sublist of all public related elements for the element. According to one aspect, the public elements include classes and interfaces, the public related elements include public superclasses and public superinterfaces, and the library is a Java™ package. According to one aspect, a method for determining a program hierarchy includes receiving an API definition file for an object-oriented library and indicating a first public element is a direct parent of a second public element when the first public element is represented in the sublist for the second public element and the first public element is not represented in the sublist for any other public element listed in the sublist for the second public element. According to another aspect, a method for detecting changes to a program hierarchy includes comparing a first program hierarchy reconstructed from a first API definition file with a second program hierarchy reconstructed form a second API definition file and indicating an error when the first program hierarchy is inconsistent with the second program hierarchy.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1A is a representation of a typical hierarchy as disclosed in a typical Application Programming Interface (API) representation.

Fig. 1B is a representation of a typical hierarchy including a submerged hierarchy.

Fig. 2A is a block diagram of a computer system suitable for implementing aspects of the present invention.

Fig. 2B is a flow diagram that illustrates a method for representing an API definition in accordance with one embodiment of the present invention.

Fig. 2C is an API representation of a hierarchy in accordance with one embodiment of the present invention.

Fig. 2D is an API representation in accordance with one embodiment of the present invention, demonstrating a change from the hierarchy as shown in Fig. 2C.

Fig. 3A is a flow diagram that illustrates determining a program hierarchy based on an API in accordance with one embodiment of the present invention.

Fig. 3B is a class diagram that illustrates the hierarchy represented by the API in Fig. 2C.

Fig. 3C is a class diagram that illustrates the hierarchy represented by the API in Fig. 3C.

Fig. 4A is a class diagram that illustrates a class hierarchy.

Fig. 4B is a class diagram that illustrates adding a class to the hierarchy of Fig. 4A.

Fig. 5A is an API definition file in accordance with one embodiment of the present invention, representing the hierarchy of Fig. 4A.

Fig. 5B is an API definition file in accordance with one embodiment of the present invention, representing the hierarchy of Fig. 4B.

Fig. 6A is a class diagram of an API definition that is extended by a client API definition.

Fig. 6B is a representation of a client API definition that references the API definition illustrated in Fig. 6A.

Fig. 6C is an API definition file in accordance with one embodiment of the present invention, representing the hierarchy in Fig. 6B.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Those of ordinary skill in the art will realize that the following description of the present invention is illustrative only. Other embodiments of the invention will readily suggest themselves to such skilled persons having the benefit of this disclosure.

This invention relates to computer systems. More particularly, the present invention relates to an API representation enabling a submerged hierarchy in an object-oriented system. The invention further relates to machine-readable media on which are stored (1) the layout parameters of the present invention and/or (2) program instructions for using the present invention in performing operations on a computer. Such media includes by way of example magnetic tape, magnetic disks, optically readable media such as CD ROMs and semiconductor memory such as PCMCIA cards. The medium may also take the form of a portable item such as a small disk, diskette or cassette. The medium may also take the form of a larger or immobile item such as a hard disk drive or a computer RAM.

According to embodiments of the present invention, API class hierarchies are specified such that submerged hierarchies are supported while sufficiently constraining particular API implementations such that each API implementation is functionally equivalent and the operations available to clients of each API implementation are exactly equivalent.

Figure 2A depicts a block diagram of a computer system 20 suitable for implementing aspects of the present invention. As shown in Fig. 2A, computer system 20 includes a bus 12 which interconnects major subsystems such as a central processor 22, a system memory 16 (typically RAM), an input/output (I/O) controller 18, an external device such as a display screen 24 via display adapter 26, serial ports 28 and 30, a keyboard 32, a fixed disk drive 34, a floppy disk drive 36 operative to receive a floppy disk 38, and a CD-ROM player 40 operative to receive a CD-ROM 42. Many other devices can be connected, such as a pointing device 44 (e.g., a mouse) connected via serial port 28 and a modem 46 connected via serial port 30. Modem 46 may provide a direct connection to a remote server via a telephone link or to the Internet via a POP (point of presence). Alternatively, a network interface adapter 48 may be used to interface to a local or wide area network using any network interface system known to those skilled in the art (e.g., Ethernet, DSL, AppleTalk™).

Many other devices or subsystems (not shown) may be connected in a similar manner. Also, it is not necessary for all of the devices shown in Fig. 2A to be present to practice the present invention, as discussed below. Furthermore, the devices and subsystems may be interconnected in different ways from that shown in Fig. 2A. The operation of a computer system such as that shown in Fig. 2A is readily known in the art and is not discussed in detail in this application, so as not to overcomplicate the present discussion. Code to implement the present invention may be operably disposed in system memory 16 or stored on storage media such as fixed disk 34 or floppy disk 38.

Turning now to Fig. 2B, a flow diagram that illustrates creating an API definition file in accordance with one embodiment of the present invention is presented. At 50, a library is received. At 52, a list of public elements in the library is created. The list of public elements includes a sublist of all public related elements. Each sublist of public related elements includes public elements that are directly related and public elements that are indirectly related. Examples of public elements that are directly and indirectly related are illustrated with reference to Fig. 2C. At 54, the list is stored.

Turning now to Fig. 2C, an API representation in accordance with one embodiment of the present invention is presented. The API representation shown in Fig. 2C can be used to represent the hierarchies illustrated in Figs. 1A and 1B. In this representation, the hierarchy of each public class is defined using a list of the public superclasses. Thus, the hierarchy for public class C2 (56) is the list of pubic superclasses (C1, Object). The hierarchy for public class C1 is (Object) and the hierarchy for public class Object (60) is the empty set, since Object (60) has no superclass. Note that private class (10) in Fig. 1B is not disclosed in the API representation of Fig. 2C.

Given the API representation of the present invention, the hierarchy can be reconstructed. For example, the hierarchy of Fig. 1A can be reconstructed from the representation illustrated in Fig. 2C as follows. Starting with public class C2 (56), the representation or list of superclasses for C2 (56) indicates C2 (56) has superclasses C1 (58) and Object (60). Therefore, either C1 (58) or Object (60) is a direct superclass of C2 (56). Since the representation for the hierarchy of C1 (58) also indicates Object (60) is a superclass of C1 (58), C1 (58) is the direct superclass of C2 (56) and Object (60) is the direct superclass of C1 (58). Determining a program hierarchy is discussed in more detail with reference to Fig. 3A.

Turning now to Fig. 3A, a method for determining a program hierarchy based on an API definition file in accordance with one embodiment of the present invention is presented. At 70, an API definition file is received. At 72, a first public element in the API definition file is received. At 74, a determination is made regarding whether the first public element is in a sublist for a second public element in the API definition file. If the public element is not in a sublist within the API definition file, the public element is not a parent of any other element in the API definition file. If the public element is in a sublist for another element, at 76, a check is made to determine whether the first public element is in a sublist for any other public element listed in the sublist of the second public element. If the answer at 76 is "Yes", at 78, an indication is made that the first public element is a direct parent of the second element. If the answer at 76 is "No", at 68, an indication is made that the first element is an indirect parent of the second element. This process continues at 72 until all public elements in the API definition file have been examined.

The API representation illustrated by Fig. 2C sufficiently constrains particular implementations from changing the order of classes in a hierarchy. This is illustrated in Figs. 2C-3C. Figure 3B shows an initial hierarchy including three classes, C2 (75), C1 (82) and Object (80). The class C2 (75) extends class C1 (82) and C1 (82) extends Object (80). Figure 2C illustrates an API representation of the hierarchy in Fig. 3B according to one embodiment of the present invention. According to one embodiment of the present invention, each class includes a list of all superclasses, both direct and indirect. Thus, class C2 (56) includes enough information regarding its superclasses to reconstruct the hierarchy represented in Fig. 3B from C2 (75) up through Object (80). Likewise, class C1 (58) includes enough information to reconstruct the hierarchy from C1 (82) up to Object (80).

Once the hierarchy is reconstructed, relationships inconsistent with the original hierarchy can be detected. Fig. 3C represents an attempt to change the initial hierarchy of Fig. 3B by switching the relationship between Object (80) and C1 (82). Figures 2C and 2D are API representations in accordance with the present invention, corresponding to Figs. 3B and 3C, respectively. Those of ordinary skill in the art will recognize that the API representation in Fig. 2C is not equal to the API representation in Fig. 2D.

Since the present invention allows the hierarchy represented by Fig. 3B to be reconstructed based on the API representation, any attempt to switch the relationship between the classes can be detected. For example, the reconstructed hierarchy created from the API representation in Fig. 2C indicates that neither C1 (58) nor C2 (56) is a superclass of Object (60), and that C2 (56) is not a superclass of C1 (58). However, the reconstructed hierarchy created from the API representation in Fig. 2D indicates that C1 (62) is a superclass of Object (64). Since the hierarchical relationships extracted from the API representations in Figs. 2C and 2D are inconsistent, an attempt to change the relationships of the classes in Fig. 3B is indicated.

Furthermore, adding or removing a publicly accessible class from a hierarchy results in a different API representation. The effect of adding a publicly accessible class is illustrated in Figs. 4A-5B. Figure 4A illustrates an initial hierarchical relationship. Figure 4B illustrates the hierarchy that results after a new publicly accessible class C3 (100) is added to the hierarchy illustrated in Fig. 4B. Figures 5A and 5B are API representations of the hierarchies in Figs. 4A and 4B, respectively. The list of superclasses for C2 (160, 180) differs in Figs. 5A and 5B. Specifically, the list of superclasses for C2 (160) in Fig. 5A includes C1 (155) and Object (150). The list of superclasses for C2 (180) in Fig. 5B includes C3 (175), as well as C1 (170) and Object (165). Since the hierarchical relationships extracted from the API representations in Figs. 5A and 5B are inconsistent, an attempt to change the relationships of the classes in Fig. 4A is indicated.

Those of ordinary skill in the art will recognize that the effect of removing a publicly accessible class may be detected in a similar manner.

h8a'

~~The API representation illustrated by Fig. 2C also allows hierarchies having submerged hierarchies such as the one illustrated by Fig. 1B to be represented without revealing any non-public or proprietary information about the hierarchy. Thus, the list of superclasses for C2 (8) is (C1, Object), the list of superclasses for C1 (14) is (Object) and the list of superclasses for Object (12) is the empty set. A client of this API definition can use this information to reconstruct the hierarchy of Fig. 1B, without revealing any information about the non-public class PrivateClass (10).~~

Since a client of an API definition only has access to public items in any particular implementation of the API definition, the existence of a submerged hierarchy does not have any impact on the client. The submerged portion of the hierarchy is non-public and therefore not available to the client. Thus, using the API representation in Fig. 2C, a client can execute with either an implementation of Fig. 1A or Fig. 1B and obtain functionally equivalent results.

A client API definition is an API definition that references another API definition. The API representation of the present invention discloses only those portions of client API definitions relevant to a client in regard to class hierarchies, without requiring complete disclosure of those referenced API definitions. This provides an additional method for keeping dependencies on referenced API definitions undisclosed, while still providing sufficient information to a client of the client API definition.

Turning now to Fig. 6A, a class diagram of an API definition that is extended by a client API definition is presented. Client API definition C4 (182) references the API definition containing C2 (184). The API definition containing C2 (184) references class C1 (186). Both class C1 (186) and class C3 (188) reference class Object (190). In this example, C4 (182) of the client API definition extends C2 (184) of the referenced API definition, and class C3 (188) of the client API definition extends class Object (190) of the referenced API definition.

128a2

~~Turning now to Fig. 6B, a representation of a hierarchy that references another hierarchy is presented. Classes Object (190) and C2 (184) are as represented in Fig. 6A.~~

Turning now to Fig. 6C, an API representation of the hierarchy of Fig. 6B in accordance with one embodiment of the present invention is presented. The API representation includes all direct and indirect superclasses for each class in the client API definition of Fig. 6A. Thus, a client of the API definition in Fig. 6C can determine complete hierarchy information from the client API (182) in Fig. 6A, without requiring complete disclosure of referenced API definition (184) of FIG. 6A.

The present invention also relates to apparatus for performing these operations. This apparatus may be specially constructed for the required purpose or it may comprise a general-purpose computer as selectively activated or reconfigured by a computer program stored in the computer. The procedures presented herein are not inherently related to a particular computer or other apparatus. Various general-purpose machines may be used with programs written in accordance with the teachings herein, or it may prove more convenient to construct more specialized apparatus to perform the required process. The required structure for a variety of these machines will appear from the description given.

While the Java™ programming language and platform are suitable for the invention, any language or platform having certain characteristics would be well suited for implementing the invention. These characteristics include type safety, pointer safety, object-oriented, dynamically linked, and virtual machine based. Not all of these characteristics need to be present in a particular implementation. In some embodiments, languages or platforms lacking one or more of these characteristics may be utilized. Also, although the invention has been illustrated showing object-by-object security, other approaches, such as class-by-class security could be utilized.

The system of the present invention may be implemented in hardware or in a computer program. Each such computer program can be stored on a storage medium or device (e.g., CD-ROM, hard disk or magnetic diskette) that is readable by a general or special purpose programmable computer for configuring and operating the computer when the storage medium device is read by the computer to perform the procedures described. The system may also be implemented as a computer-readable storage medium, configured with a computer program, where the storage medium so configured causes a computer to operate in a specific and predefined manner.

According to a presently preferred embodiment, the present invention may be implemented in software or firmware, as well as in programmable gate array devices, Application Specific Integrated Circuits (ASICs), and other hardware.

Thus, a novel method for representing an API has been described. While embodiments and applications of this invention have been shown and described, it would be apparent to those skilled in the art having the benefit of this disclosure that many more modifications than mentioned above are possible without departing from the inventive concepts herein. The invention, therefore, is not to be restricted except in the spirit of the appended claims.